B58: MIPS Tetris - Final Design Document
Zohair Syed

Overview - 256 x 256 pixels, 8 units; works on Mars and Saturn
Features - Total of 5 easy and 3 hard

Memory Layout

- Stores the display framebuffer/keyboard
- Stores the *board* (keeps track of each cell in the grid and whether it has a block committed to it or not - holds 0/1 booleans)
- Stores a list of 'active' states
    - Current row/column, column, orientation, rotation_block, next_piece
- Stores UI information
    - Including 7 tetrominos and their rotations
    - Static UI elements (letters, digits 0-9 for scoring, 'boxes' for previews)
- *Gravity* threshold/constants along with *hold* constants/booleans

Routines/Main Flow

The program starts by calling **main**. Main initializes the framebuffer, keyboard and the list of active states above. It then moves forward to **init_game,** which initializes and draws the static UI. Then the program goes into the main **game_loop,** which polls for the user's input and breaks into a switch statement to decide the next course of action. Possible inputs here include WASD for rotation, left, down, and right. Additionally, C is used to *'hold/save'* a piece for later. Once the user does an input, the program goes into the according block (ex: **move_right, move_down, try_rotate**).

These functions have similar designs. They use the input to change the active piece, detect collisions with **can_place_piece** and the *board* array and then go back to **game_loop** to be drawn with **start_drawing**.

On downwards movements, there is some more logic:

- Pieces are 'locked' to the grid with **lock_piece** if they can't go more down
- Line clearing is checked with **clear_lines**
    - If cleared, scores are updated with **update_score** and are displayed
    - Displaying scores is done with **draw_ui_element**
    - Each line clear also increases the *gravity* (makes pieces fall down faster)
- Finally new pieces are determined using **spawn_piece**

Additionally, **try_hold**, **try_rotate**, and **do_gravity** have similar paths that conclude and end up back to the **game_loop** to continue. In particular, gravity progression is calculated as countdown = max(initial - 1000 * lines cleared, 5000). If no moves are possible, the program exits.

Structures

- Currently, tetrominos are stored as 4x4 grids consisting of 16 elements (4 bytes)
- 4 copies of each tetronimo exist for each rotation (90, 180, 270 degrees)
- Colours and addresses are currently stored as bytes/words in memory
- X_piece_rotations is a struct with 4 elements (all words)
    - Each word is a pointer to one of X's rotated tetronimo pieces
- Static elements and UI are stored as multiple words of 4 bytes representing rows